Finding Boundary Shape Matching Relationships in Spatial Data

Edwin M. Knorr, Raymond T. Ng*, and David L. Shilvock

Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4
Canada

Abstract. This paper considers a new kind of knowledge discovery among spatial objects—namely that of partial boundary shape matching. Our focus is on mining spatial data, whereby many objects called *features* (represented as polygons) are compared with one or more point sets called *clusters*. The research described has practical application in such domains as Geographic Information Systems, in which a cluster of points (possibly created by an SQL query) is compared to many natural or man-made features to detect partial or total matches of the facing boundaries of the cluster and feature. We begin by using an alpha-shape to characterize the shape of an arbitrary cluster of points, thus producing a set of edges denoting the cluster's boundary. We then provide an approach for detecting a boundary shape match between the facing curves of the cluster and feature, and show how to quantify the value of the match. Optimizations and experimental results are also provided. We also describe several orientation strategies yielding significant performance enhancements. Finally, we show how top-k matches can be computed efficiently.
Keywords: spatial knowledge analysis and discovery, pattern matching, GIS

1 Introduction

Knowledge discovery is defined as the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [5]. The purpose of spatial data mining is to efficiently discover relationships or patterns in such spatial data as maps, photographs, and satellite images. Efficiency is a recurring theme in this paper. Although many researchers have investigated relationships that exist among *relational* data [1,2,6,7], a few others have gone beyond the relational model in order to discover relationships in spatial domains [9–12]. To the best of our knowledge, Lu, Han and Ooi were the first ones to develop spatial data mining techniques [10]. They proposed two algorithms that extract high-level relationships between spatial and nonspatial attributes in spatial databases. However, both algorithms require spatial concept hierarchies to be provided.

* Person handling correspondence. Email: rng@cs.ubc.ca

This is a serious restriction, because for most applications, it is very difficult to know a priori which hierarchies would be the most appropriate. To overcome this problem, Ng and Han developed the CLARANS algorithm which can find spatial clustering structures implicit in the data [11]. Without relying on any spatial hierarchy, CLARANS may discover, for example, that the most expensive housing units in Vancouver can be grouped spatially into a few clusters. Ester, Kriegel and Xu proposed several focusing techniques that can be used to further optimize the performance of CLARANS [4].

While CLARANS, and for that matter any clustering algorithm, is effective in answering the question of *what* the clusters are, it fails to answer the question, which is more interesting from a knowledge discovery point of view, of *why* the clusters are there spatially. But since we believe that in general it may be too much to expect a computer to be able to pinpoint exactly why a cluster is there spatially, we aim to develop algorithms that can answer a weaker form of the question—namely, what the characteristics and patterns of the clusters are. In [9], we developed efficient algorithms for identifying characteristics based on features that are near to, or far from, the clusters. In [12], we studied how to extract common and discriminating characteristics of clusters from thematic maps. In this paper, we consider another kind of characteristic—namely boundary shape matching relationships among 2-D spatial objects.

1.1 Problem Definition and Contributions

Below, we define boundary shape matching (BSM) and motivate why it is of interest to spatial data analysis and mining. Fig. 1 shows a cluster of 2-D points (x's representing houses), and a man-made feature (a golf course) described by a simple polygon. Note that parts of their boundaries match. We call this a (partial) BSM relationship between the cluster and the golf course. This simple example shows the cluster and the feature adjacent to each other, but in general we do not require proximity for BSM. In this paper, we refer to the point set as a *cluster* and to the geographic entities in a set of maps as *features*. The problem we study in this paper is:

Given a cluster, we find the top-k features, as well as the parts of the matched boundaries, that have the strongest BSM relationships with the cluster.

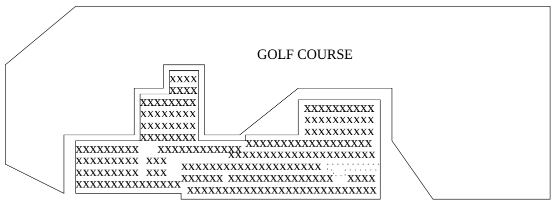


Fig. 1. Boundary shape matching between a housing cluster and a golf course

BSM is useful for spatial data analysis and mining because it can identify some prominent relationships. For example, if part of a housing cluster follows the contour of a golf course, lake, or park, it makes sense to report this fact. In the real estate market, there can be a huge price differential between houses that are across the street from (or simply near) a golf course, versus houses that actually follow the contour (e.g., fairways) of the golf course. (See [9] for a way of analyzing *proximity* relationships among clusters and features.) Note that a relationship can exist even if the cluster and the feature are relatively far apart. For example, if a cluster of expensive houses is a kilometre away from the shoreline, yet follows the shoreline reasonably well, one would suspect a possible relationship. Thus, if the cluster were on a slope, one might be led to believe that some or many of the houses in the cluster yield a splendid view. Relationships such as those described above may be of interest to such groups as the real estate community, Geographic Information Systems (GIS) analysts, demographers, and planners. Although our testbed is GIS, other spatial domains also apply. To show that BSM can be applied concretely, this paper uses real estate examples, and regularly and irregularly shaped geographic features in the city of Vancouver. As a preview, in this paper we provide answers to the following questions key to BSM:

- What is the "boundary" or the "shape" of a collection of points? Our notion of shapes is based on alpha-shapes introduced in [3,8]. Sect. 2 will show how alpha-shapes are adapted for and used in BSM.
- What is a "good fit" between two boundaries? In the ideal case, two boundaries could fit like two adjacent pieces in a jigsaw puzzle. More specifically, if each boundary is represented by a sequence of connected line segments, the characteristics of a good fit are that the peaks and the valleys line up, and that there is a constant separation distance between the two boundaries. However, there is the complication of the orientations of the boundaries. Our solution to this complication is to try many orientations in an intelligent manner. Sect. 3 will define all of these ideas more vigourously.
- How can we quantify the strength of a BSM relationship? Our ranking function takes into account the goodness of fit, partial matching boundaries, the length of the matching sub-boundaries, and a few complications such as hidden segments. Sect. 3 will provide the details.
- How can we efficiently identify the features with the top-k strongest BSM relationships? Our algorithm is based on filtering. In terms of performance, for a pool of 326 features, our algorithm can identify the top-10 features and their matching sub-boundaries in less than 3 seconds of CPU time. Sect. 4 will explain the ideas and show experimental results.

2 Defining Shapes of Clusters by Alpha-Shapes

As we have indicated, BSM will be performed on clusters and features. Each feature is defined by a sequence of points that make up a polygon. Most maps accessible by computer contain features described by an ordered sequence of (x, y) vertices defining the endpoints of a closed curve of line segments (edges). A cluster, on the other hand, is simply a set of unordered points with no specific boundary, perhaps having been produced in response to an SQL query. While clustering algorithms such as CLARANS [4,11] show how to identify one or more clusters from a set of points, no specific boundaries are associated with those clusters. This leads us to our first question: What is the boundary and shape of a cluster? Given a (dense) cluster of points, most humans can easily visualize a polygon whose shape characterizes a cluster. While the shape of such a polygon may be intuitive to humans, it is undefined to machines.

2.1 Basics of Alpha-Shapes

We begin by defining the shape of a cluster as a *convex shape*, which is simply a closed boundary of a set of points such that the line segment connecting any two points in the set always lies on or interior to the boundary. Among convex shapes, the *convex hull* is the most natural choice, because it is the unique, minimum bounding convex shape enclosing a set of points [13]. Convex hulls are a powerful construct for many spatial applications, including data mining [9]; however, they are not very good at providing detailed shape information about nonconvex spatial objects. For example, if we were limited to using a convex hull to describe the cluster in Fig. 1, most of the interesting aspects of the shape would be lost, and BSM would be difficult, if not impossible.

Because the shape of a cluster may be nonconvex, and because convex hulls are too crude for effective BSM, we turn to a generalization of convex hulls, namely *alpha-shapes* [3,8]. While a cluster has a unique convex hull, there is a spectrum of alpha-shapes to consider, each alpha-shape being defined by some α-value. A zero α-value gives the convex hull, positive α-values produce cruder convex shapes for a given set of points, and negative α-values provide finer nonconvex resolutions. The boundary of the cluster in Fig. 1 is the result of a negative α-value. The time complexity for constructing an alpha-shape is the same as that for constructing the convex hull. The complexity is $O(n \log n)$, where n is the number of points in the set.

2.2 Applying Alpha-Shapes to BSM

While certain members of a family of alpha-shapes can describe the intuitive shape of a nonconvex cluster, it is unclear what α-value to use in general, because the choice can vary greatly from cluster to cluster. Furthermore, a fine resolution produced by a large negative α-value may be too fine in the sense that there may be severe, narrow cavities or fissures in the resulting polygon. To illustrate this point, consider Fig. 1 again. In the right side of the cluster, we see a fissure (represented by a dotted area) which may be the result of certain negative α-values. Many such fissures may exist. For BSM, we want alpha-shapes that are neither too crude, nor too fine. In general, it is very difficult to determine the best α-value.

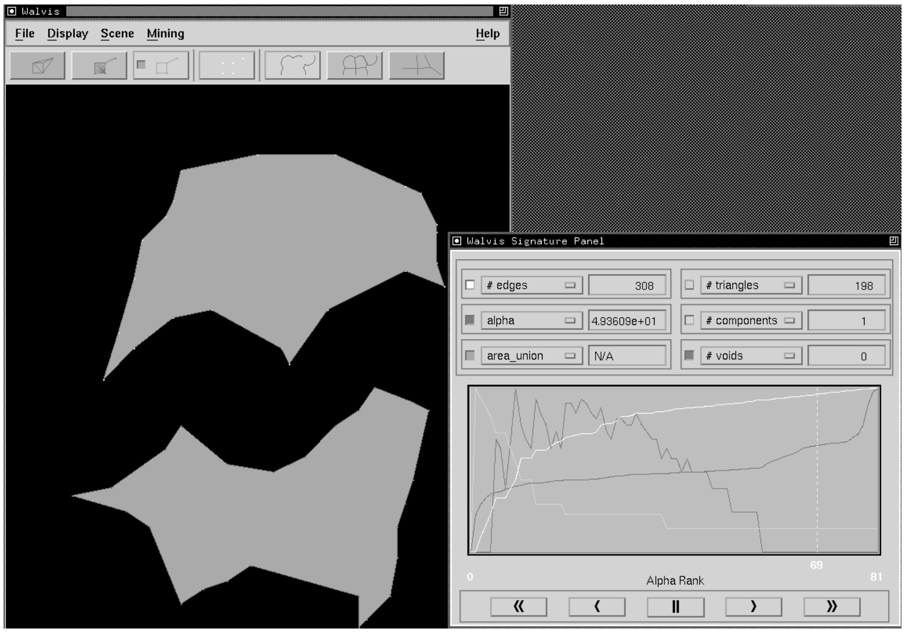


Fig. 2. Matching two alpha-shapes

To play with alpha-shapes empirically, we developed a prototype system, part of which is based on the software called "Shape2D-v1.1" from the National Center for Supercomputing Applications and the Department of Computer Science at the University of Illinois at Urbana-Champaign.[1] Fig. 2 shows a screen session of part of our prototype system. In this session, there are two windows. The left window shows two alpha-shapes brought together for matching. The right window can be used to adjust the alpha-shapes. This can be done by pressing the appropriate buttons at the bottom of the right window. The top part of the window shows how various attributes of the alpha-shape (such as the number of edges, connected components, and voids (holes) in the shape) are changed when the alpha rank is adjusted. The middle part of the window gives a graphical display of the values of these attributes.

We ran many experiments to show how different α-values can affect the shape of a cluster. We observed that an alpha-shape can become disconnected and can change significantly even for small changes in α. For the point sets that we tested, we observed that the best α-values for our application were those for which the number of voids in the shape was zero, and the number of connected

[1] This package uses an α-value that differs from the original definition [3]. The package's α-value is the reciprocal of the original α-value. The range of allowable α-values for this package results in a family of shapes ranging from the original, unconnected point set, up to and including the convex hull. Unless indicated otherwise, we use the original definition throughout this paper.

components was one. Even so, a single-component, zero-void alpha-shape can occur for several ranges of α-values. We chose the α-value for a given cluster to be the average of the midpoints of all ranges that yield one component and no void. Using this value resulted in a reasonable alpha-shape for the cluster. Its boundary is thus defined by an ordered sequence of vertices (and hence, line segments), which we use in the BSM algorithm described next.

3 Quantifying BSM Relationships

In this section, we describe how BSM relationships are computed and quantified. We explain the notions of facing and difference curves, give scoring functions for ranking BSM relationships, and describe how the axis of orientation can be selected intelligently.

3.1 Reducing Two Object Boundaries to Their Facing Curves

To develop an effective BSM algorithm, many questions have to be answered. The first question concerns which parts of the boundaries are to be considered. Regardless of the relative positions of the cluster and the feature on a map, it makes sense (particularly for GIS applications) to consider only those parts of the boundaries that face each other. We use outer common tangent lines to delimit the facing curves, as shown in Fig. 3. Intuitively, two chains of line segments face each other in the sense that the endpoints (i.e., vertices touching the tangent lines) and at least one vertex between the endpoints (in the nontrivial case) can "see" some vertex in the other chain. Formally, if x is a vertex in polygon A, and y is a vertex in polygon B, then we say that x and y see each other if the line segment joining x and y does not intersect the interior of either polygon A or polygon B. This definition ignores any objects that may lie between the two polygons.

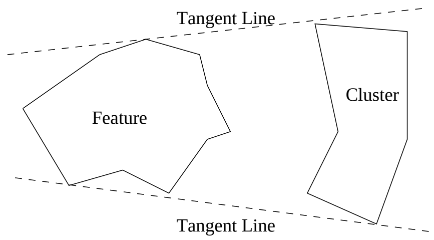


Fig. 3. Identifying facing curves by using tangent lines

A keen reader would observe that for a cluster-feature pair, there may be an infinite number of facing curves and tangent lines, if the feature were to rotate around the cluster. Because we are dealing with features and clusters that have fixed spatial locations, we do not consider the rotation of a feature around a cluster in this paper.[2] Hence, our BSM problem is reduced to determining how well two facing curves match.

3.2 Identifying the Difference Curve

The next question is how to determine whether two shapes or facing curves match. One obvious approach is to translate the curves until they touch, and then calculate the area between them. The smaller the area between them, the greater the likelihood of a match. This naive approach has several problems. First of all, it is biased toward shorter curves. For example, a pair of long curves may match well, but receive a poorer score than a pair of short curves that do not fit quite as well. Secondly, a small local change in one curve (for example, a long, sharp protrusion) can seriously affect the score. So, while we strive for efficiency in spatial data mining, and are often satisfied with approximate answers, those approximate answers still need to be accurate enough to identify good matches rather than weak ones.

The problems described above lead us to consider the notion of "puzzle fit" between two facing curves. Intuitively, if the two facing curves were the appropriate boundaries of two adjacent pieces of a jigsaw puzzle, they would fit snugly, in the sense that the peaks and valleys line up, and that there is a zero separation distance between the two curves. For the kinds of spatial analyses we wish to support, features need not be physically adjacent to a cluster. Thus, we generalize from a zero separation distance to a constant separation distance in the ideal case. We examine the *change* in the distance between the curves as we traverse an appropriate *axis of orientation*. An axis of orientation or "line of scrimmage" separates the curves so that the cluster is on one side of the line, and the feature is on the other. Separating the curves in this manner helps prevent situations in which we have to deal with overlapping curves. As we traverse the relevant part of the axis of orientation, we calculate the separation between the curves. The separation gives rise to a *difference curve*, shown in Fig. 4, which is created by plotting the distance between the curves as the axis is traversed from vertex to vertex. A change in the slope of the difference curve results only when a vertex from either curve is encountered having the same w-coordinate (where w is the axis of orientation). If there are n vertices in the cluster's curve and m vertices in the feature's curve, then $O(n+m)$ distance calculations are required for this scheme.

3.3 Quantifying the Degree of Partial Match

In Fig. 4, if V_A is the set of vertices belonging to Curve A, and V_B is the set of vertices belonging to Curve B, then let $V = V_A \cup V_B$, such that vertices in V are in

[2] The work described in this paper can be extended to include rotations. Practical applications include matching pottery fragments, matching fragmented manuscripts, and tracking icefloes.

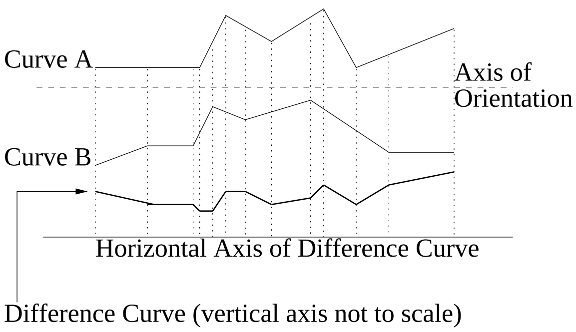


Fig. 4. Computing the difference curve

increasing order along the axis of orientation. Let $dist_i$ be the distance between the two curves (perpendicular to the axis of orientation) at vertex $V_i \in V$, and let L_i be the distance along the axis of orientation between vertices V_i and V_{i+1}. Define θ_i to be the absolute value of the slope of the difference curve (piecewise), that is, $\theta_i = \mid dist_{i+1} - dist_i \mid / L_i$. Finally, let Δ_g define the tolerance or maximum acceptable slope of a segment of the difference curve. The following equation shows one simple way to calculate the goodness of fit, where s is the number of segments in the difference curve:

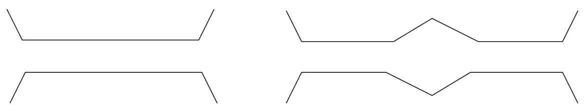
$$\text{score} = \sum_{i=1}^{s} \begin{cases} L_i(\Delta_g - \theta_i) & \text{if } \theta_i < \Delta_g \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

At first glance, (1) seems to satisfy our basic notion of fairness in assigning higher scores to flatter and/or longer difference curve segments. If $\theta_i < \Delta_g$, a larger L_i increases the score, and the closer θ_i is to zero, the larger the score.

Example 1. Let us represent the i-th line segment of a difference curve by the pair $\langle L_i, \theta_i \rangle$. Consider the difference curve: $[\langle 2,5 \rangle, \langle 20, 0.5 \rangle, \langle 2,5 \rangle, \langle 2,2 \rangle, \langle 10, 0.5 \rangle, \langle 2,5 \rangle]$. If Δ_g is set to 1, (1) only assigns positive scores to the two line segments with $\theta_i = 0.5$. The score is 20 * (1 - 0.5) + 10 * (1 - 0.5) = 15.

To evaluate the effectiveness of the above scoring function, we ran it against a set of geographic features of Vancouver. We observed that the resulting score often did not highlight the strength or quality of an excellent match. In other words, there was no strong distinction between excellent matches and middling matches. This was further complicated by the presence of bad segments within a curve. For example, Fig. 5 shows two pairs of curves that may receive identical scores using the existing linear equation—depending, of course, on the value of Δ_g.

The linear equation presented in (1) was changed in several ways. First of all, we now count both good and bad segments. Good segments are those for which $\theta_i < \Delta_g$; bad segments are those for which $\theta_i > \Delta_b$. Secondly, we show



(a) (b)

Fig. 5. Two pairs of curves that may receive identical scores using the linear equation

favouritism toward flatter difference curves. To achieve this goal, we use an exponential function instead of a linear function. Thirdly, because the difference between a good segment and a bad segment can be highly subjective, we allow for a buffer zone, whereby we assign neither a penalty nor a premium to segments falling into the range $\Delta_g \leq \theta_i \leq \Delta_b$. These three improvements produce the final form of our equation:

$$\text{score} = \sum_{i=1}^{s} \begin{cases} L_i \mathrm{e}^{(\Delta_g - \theta_i)c} & \text{if } \theta_i < \Delta_g \\ -L_i \mathrm{e}^{\min[1,\ (\theta_i - \Delta_b)/c]} & \text{if } \theta_i > \Delta_b \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The ceiling $L_i\mathrm{e}$ is used to bound the penalty for bad segments; otherwise, if θ_i greatly exceeded Δ_b, the penalty could be too severe (even for a relatively small L_i), and weaken the overall score for what might otherwise be a very good match. Furthermore, a constant $c > 1$ is used to help normalize the positive and negative terms because Δ_g and Δ_b are likely to be small compared to θ_i. Without c (and the ceiling), we would typically get small positive terms and large negative ones, making partial matches difficult to detect. We tried numerous values for Δ_g, Δ_b, and c, and found that $\Delta_g = 0.1$, $\Delta_b = 0.3$, and $c = 10$ gave reasonably good results—roughly comparable to the results a human might obtain from a quick visual observation of two facing curves. Although we cannot claim that these are the ideal values for a particular application, our experimental results suggest that these are good heuristics for our testbed.

Equation (2) readily supports the identification of *partial* matches within a pair of facing curves, which is particularly important for this type of knowledge discovery application, because it may be quite unlikely that two facing curves yield a perfect match. We also noted that some curves match quite well except for the final segment at either end of a curve's chain (that is, those segments touching the tangent lines). It seems unfair to penalize an otherwise good curve that just happens to "tail off" at its extremities. For example, although the rightmost segment of Curve B in Fig. 6 falls within the tangent lines, it obviously does not contribute to curve matching. We decided not to count these bad areas in the score. Bad areas lying between good areas are, of course, counted. To be more precise, we can reduce a difference curve to a sequence of the appropriate length from the alphabet {B,G,N}, where B and G stand for bad and good segments respectively, and N stands for neutral segments (i.e., neither a penalty nor a premium is applied). The scoring function presented above quantifies the degree of partial match according to the largest subsequence with G at both ends.[3]

Example 2. Let us consider the difference curve introduced in Example 1. It is: $[\langle 2,5\rangle, \langle 20,0.5\rangle, \langle 2,5\rangle, \langle 2,2\rangle, \langle 10,0.5\rangle, \langle 2,5\rangle]$. If Δ_g is set to 1 and Δ_b is set to 4, the difference curve can be reduced to the sequence BGBNGB. Equation (2) then applies only to the subsequence GBNG. More specifically, if the normalization constant c is set to 2, the score is given by: $20 * \mathrm{e}^{(1-0.5)*2} - 2 * \mathrm{e}^{(5-4)/2} + 10 * \mathrm{e}^{(1-0.5)*2} = 30\mathrm{e} - 2\mathrm{e}^{0.5} \approx 78$.

3.4 Robustness: Trivial Matches and Hidden Segments

Our testbed included a lot of rectangular features such as schools, playgrounds, parks, and shopping centres. We observed that many of the best scores resulted from trivial matches involving one-line segments. To understand why this is a problem, consider a rectangular shopping centre located directly north of a cluster. If one segment in the cluster (especially a long segment) is approximately parallel to a line segment in the feature, then the resulting score can be quite high. We cannot claim, however, that this constitutes a relationship of interest. On the other hand, if many line segments match, such as when an irregularly shaped cluster matches an irregularly shaped feature (e.g., a cluster following a jagged shoreline), it makes sense to report this match as a relationship of interest. We therefore apply the following rule. After determining the tangent lines, if the number of vertices in either the cluster's curve or the feature's curve is ≤ 3, we simply ignore the feature, and move on to the next one. We chose the value 3 because many features in an urban area are rectangular (or nearly rectangular). If an entire curve consists of a corner (that is, three adjacent vertices), we simply skip the calculation, thereby avoiding many trivial matches. Our experiments confirm that this is a useful heuristic.

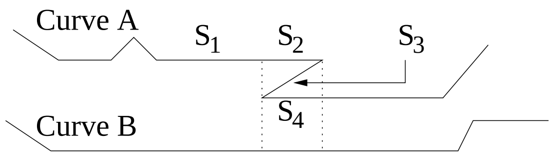


Fig. 6. Curve with hidden segments

Another issue that we had to address concerned overcounting. We encountered this situation when dealing with hidden curves, that is, curves that backtrack. When calculating the score for the pair of curves shown in Fig. 6, we ignore

[3] We can use a variant of this scoring function, which returns the highest score of *any* subsequence beginning with, and ending with, a G; however, for the sake of performance, and for returning longer, potential candidates (matches), we use (2), as described above.

the hidden parts of Curve A, which are the line segments in the diagram labelled S_2 and S_3, but we retain S_1 and S_4. If we were to include S_2, for example, an undeservingly high score would result from counting part of Curve A twice.

3.5 Varying the Axis of Orientation Intelligently

As described in Sect. 3.2, a key aspect of constructing the difference curve is the choice of axis of orientation. Using Fig. 7, we describe how to choose an appropriate axis of orientation. First, we calculate a bounding box for each of the facing curves, and then we construct tangent lines to these bounding boxes. The tangent lines are the dotted lines shown. We define our initial axis of orientation to lie in the middle of these two extremes, and we can rotate the axis through any angle ϕ, if ϕ does not go beyond a tangent line. For any angle ϕ, we calculate the separation according to where each curve intersects a line perpendicular to the corresponding axis.

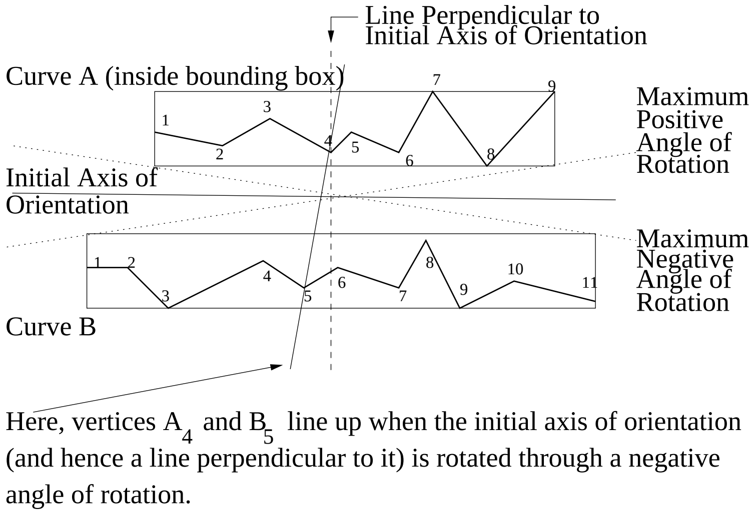


Fig. 7. Axis of orientation

Different axes of orientation result in different matches between peaks and valleys. For example, if ϕ=0, vertices A_6 and B_7 appear to line up or match, but this is not the situation with vertices A_4 and B_5, as shown by the vertical dashed line. On the other hand, if we rotate the angle of orientation through a negative value of ϕ not exceeding the limit posed by the corresponding tangent line, we see that vertices A_4 and B_5 line up. Different values of ϕ can have considerable effect in determining how well the two facing curves match. Note that different angles of rotation may be more suitable for certain pairs of vertices. There is no way of knowing in advance which angle of ϕ is best. One approach is to take a brute-force approach by exhaustively trying the entire range of allowable ϕ values using a small linear increment, and then choosing the axis that produces the highest score. Although this approach works, it does not scale very well. Efficiency is important because we want to try as many features as possible.

During our analysis of the brute-force results, we plotted each score against its corresponding value of ϕ. We noticed that many graphs contained prominent patterns. For example, a graph often contained several spikes of high scores that stood out from the rest of the graph. This suggested that we might be able to exploit these patterns, and avoid having to test a wide range of ϕ values. Several alternatives were tested:

1. Search for the first local peak.[4] Begin by calculating the score when ϕ=0, and then either increase or decrease ϕ, depending on which direction produces a score greater than or equal to the previous one. After the direction is determined, continue increasing (or decreasing) ϕ until the new score is less than the previous score, or the limit of the range is reached. At that point, report the peak value as the score for that pair of curves.
2. Divide the range of allowable ϕ values into three equal parts, search for a local peak in each part, and report the highest of the three scores.
3. Use an adaptive technique. Begin by traversing the range of allowable ϕ values using a coarse increment (e.g., 1/30 of the total range). While progressing through these coarse increments, retain the best score and its corresponding ϕ. After calculating the score for a subsequent ϕ_k, if the slope of the graph changes from increasing to decreasing, this indicates that a peak occurs somewhere between ϕ_{k-2} and ϕ_k. In this case, traverse the range defined by ϕ_{k-2} and ϕ_k again but using a finer increment (e.g., 1/80 of the total range) to see if a better score can be obtained. Following this, continue traversing the original range with the coarse increment.

Table 1. Relative effectiveness and efficiency of various orientation strategies

	Relative Effectiveness	Relative Efficiency
Brute-Force	100	100
1 Local Peak	92	58
3 Local Peaks	96	83
Adaptive	100	52

Table 1 shows a representative comparison of the four approaches, with the brute-force method being the baseline. The 1-local-peak method is faster but produces results with lower quality. The 3-local-peaks method is in between

[4] Here, we are not referring to a local peak in one of the facing curves, but rather a local peak in the graph of score versus ϕ.

the 1-local-peak method and the brute force approach in both efficiency and effectiveness. The adaptive method is the clear winner. It has an effectiveness of 100% which means that the score returned by this method is equal to that returned by the brute-force method. But the CPU time used is only 52% of that used by the brute-force method.

Our testbed had 326 real-life features in it, with a broad mix of convex, nonconvex, large, small, regularly shaped, and irregularly shaped features of the city of Vancouver. Running the adaptive technique on a Sun SPARCstation LX computer with 24 MB of main memory, we could examine a cluster-feature pair in approximately 0.02 seconds, on average.

3.6 Summary

Having described the various pieces necessary for extracting and quantifying the BSM relationship between a cluster and a feature, we can now stitch all these pieces together to form the algorithm shown in Fig. 8. Throughout the rest of this paper, we call the score computed by Procedure BSM the *BSM score* of the feature.

```
Procedure BSM(Cluster, Feature)
{
  1.1 Set the current maximum score to 0;
  1.2 Extract the facing curves as in Sect. 3.1;
  1.3 Use the adaptive approach and iterate for each axis of orientation
      (switching between coarse and fine increments as shown in Sect. 3.5)
  {
     1.3.1 Identify the difference curve as in Sect. 3.2;
     1.3.2 Check for trivial matches and hidden segments as in Sect. 3.4;
     1.3.3 Compute the score based on (2) in Sect. 3.3;
     1.3.4 If the computed score is higher than the current maximum score,
           replace the current maximum score with the computed score;
  }
  1.4 Output the current maximum score;
}
```

Fig. 8. Algorithm for computing the BSM score of a feature

In summary, here in Sect. 3, we focused on how to identify and quantify the BSM relationship between a given cluster and feature. In the next section, we turn our attention to efficiently finding the top-k features that have the strongest BSM relationships with a given cluster.

We conclude this section by noting that in computer vision and robotics, there are techniques that deal with 2-D and 3-D curve matching (e.g., [14]), but to the best of our knowledge, the techniques are based on physical constraints and do not focus on fast, approximate computation of similarity scores to possible matching objects. As for related work in computational geometry, we are not aware of any work on similarity curve matching based on alpha-shapes.

4 Finding the Top-k Features Efficiently

4.1 SDD vs SDI Rankings

Although a case can be made for applying Procedure BSM only to features lying within a very short distance of the cluster, we did not restrict ourselves in this way because there are applications in which some very interesting items of knowledge discovery are independent of distance. For example, consider a housing cluster that is located on a hill some distance from the shoreline, and which follows the outline of a sheltered bay. We would like to highlight such a spatial relationship. We refer to this approach as the *separation distance independent (SDI)* approach. Under this approach, the top-k features are those that have the k-highest BSM scores.

While there are applications that find the SDI approach appropriate, there are other applications that may prefer to rank features inversely proportional to the separation distance between the cluster and features. That is to say, if the BSM scores of two features are identical, then the one closer to the cluster is ranked higher than the one further away. We refer to this approach as the *separation distance dependent (SDD)* approach. In particular, in this paper we define the SDD score to be the BSM score divided by the separation distance. Under this ranking system, the top-k features are those that have the k-highest SDD scores.

4.2 Efficiency Optimization: Filtering and Upper Bounds

Regardless of whether or not the SDI or SDD approach is used, one way to compute the top-k features is to apply Procedure BSM to every feature one-by-one, and sort the scores (divided by the separation distance, if SDD scores are used) to find the k highest ones. Using the experimental performance figures given in the previous section, for 326 features, the total time taken would be around $0.02 * 326 \approx 6.5$ seconds.

To optimize the performance of finding the top-k features, we use filtering and upper bounds. More specifically, the algorithm keeps track of the k highest scores encountered so far in the search. For the current feature to be tested, instead of applying Procedure BSM directly, which involves a lot of operations as described in Sect. 3, the algorithm applies some lightweight calculation of the best possible score—that is, providing an upper bound on the actual score. If the best possible score attainable for a particular cluster-feature pair is already lower than all the current k highest scores, Procedure BSM need not be applied. Otherwise, the procedure is used to compute the actual, exact score. In this way, as the search for the top-k features goes on, it is expected that more and more features can be screened out by upper bounding and filtering, and many invocations of Procedure BSM can be saved.

The key question is the choice of the upper bound. If the upper bound is too loose, then very few features can be screened away, in which case the entire effort of filtering is pointless. Tight upper bounds are preferrable—if they can be computed easily. If the upper bounds are as complicated to compute as Procedure BSM itself, then again filtering is pointless. Thus, there is the tradeoff between the effectiveness and efficiency of upper bounding.

In our study, we used the following upper bounds. For the SDI approach, we upper bounded the BSM score by the equation $\sum L_i e^{c\Delta_g}$. It is obvious that the score defined by (2) cannot exceed the above upper bound. Notice that in this upper bound, only L_i's are dependent on the feature, as both c and Δ_g are design parameters. In terms of computational effort, the upper bound only requires the extraction of the facing curves, but not any computation and iteration in Procedure BSM that requires the axis of orientation and the difference curve.

The above upper bound on the BSM score applies to the SDD approach as well. But for the SDD approach, more can be done. In particular, we tried the following two optimizations:

– We lower bounded the real separation distance between two curves by the separation between their bounding boxes. This separation distance is easy to compute ($O(1)$ time) after finding the bounding boxes in linear time. Again, this lower bound can be computed based on the facing curves only, but not the axis of orientation and the difference curve.
– Regardless of whether we used the bounding box distance or the exact separation distance, we sorted that distance and tested the features in ascending order. This is because as the distance becomes larger and larger, the chance that a feature will make the top-k SDD list decreases. This makes filtering more effective.

4.3 Experimental Evaluation

To empirically evaluate our upper bounding techniques, we used the dataset introduced at the end of Sect. 3.5. Table 2 shows the run times (in seconds) when applying various optimizations to find the top-10 features. These BSM times include all the computations described in Sects. 3.1 to 4.2, but exclude the time needed to compute the cluster and its alpha-shape. The number in parentheses shows the number of features filtered away, out of a total of 326 features.

If no optimization was applied, the time taken was 6.5 seconds and no feature was filtered out. Once the BSM scores were upper bounded, the time taken reduced to 64%, or 4.2 seconds, and almost half of the features, 150 to be exact, did not require the invocation of Procedure BSM. This indicates that even the SDI approach, which does not involve separation distance, can benefit by filtering and upper bounding.

For the SDD approach, the optimizations are even more pronounced. The third row in Table 2 indicates that by just lower bounding by the bounding box distance, the time taken reduced to 60%, or 3.9 seconds, and 184 features were

Table 2. Efficiently finding the top-k features

	seconds (features filtered out)
no optimization upper bounding BSM scores	6.5 (0) 4.2 (150)
lower bounding by bounding box distance	3.9 (184)
upper bounding BSM scores + sorting separation distance	3.0 (248)
upper bounding BSM scores + lower bounding by bounding box distance + sorting bounding box distance	2.8 (236)

filtered out. This demonstrates that the bounding box distance optimization is effective.

Furthermore, by comparing the second and fourth rows in Table 2, it is clear that sorting the distance (in this case, the real separation distance) is also useful. The sorting, in conjunction with upper bounding BSM scores, reduced the run time to 46%, or 3.0 seconds, and filtered out 248 features.

The last row in Table 2 shows the performance figures when all the optimizations described above were applied. In this case, the run time was reduced to 43% or 2.8 seconds, and 236 features were filtered away. An interesting point can be made by comparing the number of filtered features in the fourth and last rows. The algorithm corresponding to the last row actually needed to invoke Procedure BSM more times than the algorithm corresponding to the fourth row. But computing bounding box distances is more efficient than computing real separation distances. The savings achieved by this optimization more than offsets the effort required by the extra features.

Fig. 9 shows a screen dump of finding top-k BSM relationships with our prototype system, which implements all the ideas described so far. The cluster and its top-10 features are displayed on a map. The bottom right corner gives the list of the top-10 features and their associated scores. If a user wants to examine the BSM relationship between a feature and the cluster more closely, the user can bring up the screen shown in Fig. 2.

5 Conclusions

We described a problem in spatial knowledge discovery involving boundary shape matching. We began by introducing alpha-shapes as a tool for finding a boundary to describe the shape of a cluster of unordered points. We then described how to perform BSM, and how to quantify the results. Several optimizations and orientation strategies were presented. Our experiments showed that our adaptive approach produces efficient and effective BSM, including partial and total matches of facing curves. Finally, we presented a few optimizations for identifying the top-k features. The optimizations appear to be effective empirically.

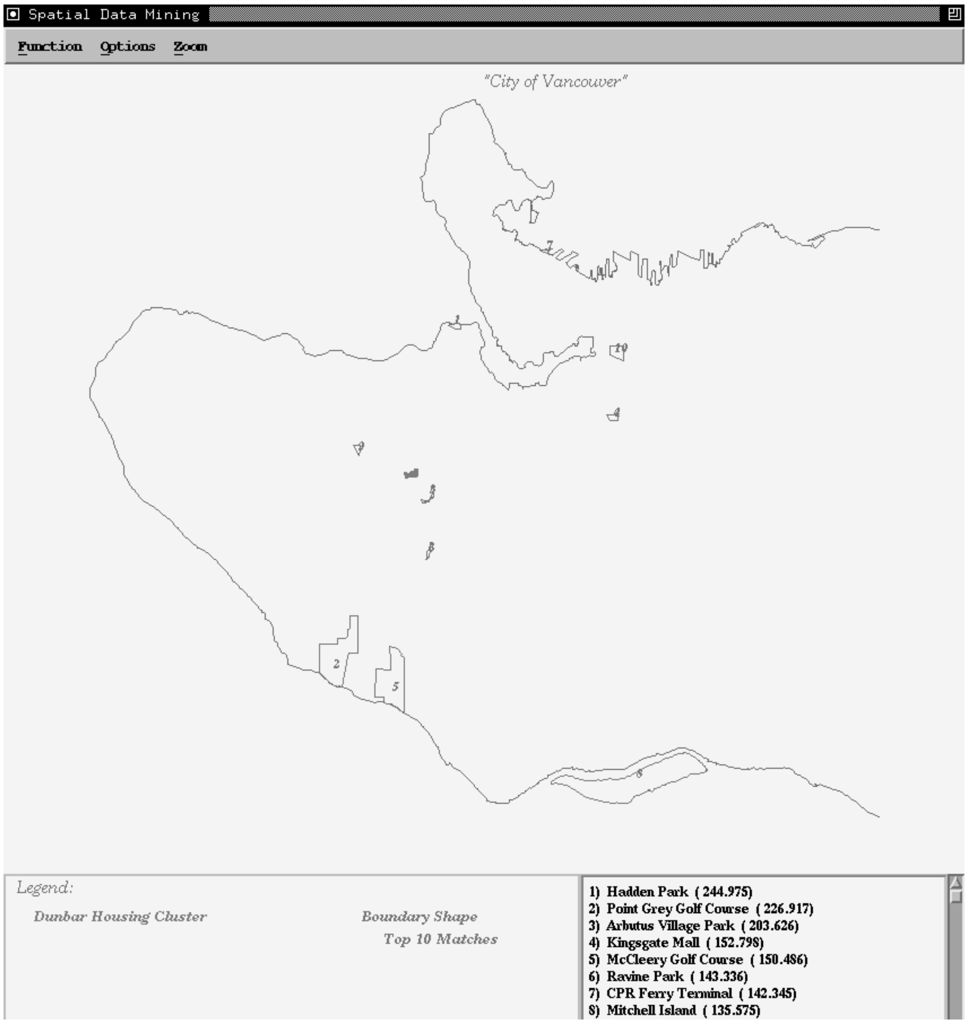


Fig. 9. Finding the top-10 BSM relationships

References

1. R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. "An Interval Classifier for Database Mining Applications", *Proceedings of the 18th VLDB Conference*, pp. 560-573, 1992.
2. R. Agrawal, T. Imielinski, and A. Swami. "Mining Association Rules between Sets of Items in Large Databases", *Proceedings of the 1993 SIGMOD Conference*, pp. 207-216, 1993.
3. H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. "On the Shape of a Set of Points in the Plane", *IEEE Transactions on Information Theory*, 29, 4, pp. 551-559, 1983.
4. M. Ester, H. Kriegel and X. Xu. "Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification", *Proceedings of the 4th International Symposium on Large Spatial Databases (SSD'95)*, pp. 67-82, 1995.
5. W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. "Knowledge Discovery in Databases: An Overview", *Knowledge Discovery in Databases*, Piatetsky-Shapiro and Frawley (eds.), AAAI/MIT Press, pp. 1-27, 1991.
6. J. Han, Y. Cai, and N. Cercone. "Knowledge Discovery in Databases: an Attribute-Oriented Approach", *Proceedings of the 18th VLDB Conference*, pp. 547-559, 1992.
7. D. Keim, H. Kriegel, and T. Seidl. "Supporting Data Mining of Large Databases by Visual Feedback Queries", *Proceedings of the 10th International Conference on Data Engineering*, pp. 302-313, 1994.
8. D. Kirkpatrick and J. Radke. "A Framework for Computational Morphology", *Computational Geometry*, G. Toussaint (ed.), The Netherlands: Elsevier Science Publishers B.V., 1985, pp. 217-248, 1985.
9. E. M. Knorr and R. T. Ng. "Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining", *IEEE Transactions on Knowledge and Data Engineering*, 8, 6, pp. 884–897, December, 1996.
10. W. Lu, J. Han, and B. C. Ooi. "Discovery of General Knowledge in Large Spatial Databases", *Proceedings of the Far East Workshop on Geographic Information Systems*, Singapore, pp. 275-289, 1993.
11. R. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining", *Proceedings of the 20th VLDB Conference*, pp. 144-155, 1994.
12. R. Ng and Y. Yu. "Discovering Strong, Common and Discriminating Characteristics of Clusters from Thematic Maps", *Proceedings of the Eleventh Annual Symposium on Geographic Information Systems*, pp. 392-394, 1997.
13. J. O'Rourke. *Computational Geometry in C*, Cambridge University Press, New York, 1994.
14. J. T. Schwartz and M. Sharir. "Identification of Partially Obscured Objects in Two and Three Dimensions by Matching Noisy Characteristic Curves", *International Journal of Robotics Research*, Vol. 6, No. 2, pp. 29-44, Summer 1987.